Efficient Moving Average and Random Walk Functions Defined in C++

A function was written in C++ to calculate the moving average and implemented in R using the cxxfunction of the RCPP package. Function arguments passed from R to C++ were a vector (x) that contained the data to be averaged, and an integer (k) that gave the size of the moving average window. The function consisted of a *for* loop that iterated through the elements of vector x, summing the elements within each sequential window, dividing by the window length, storing the result in a vector that was finally returned to R at the end of the loop. A *while* loop controlled the progress of the function, ending when the last window length in the vector passed from R is reached.

Declare and define variables in C++

The first step in this code was to convert the data types passed from R to C++(Rcpp) data types. These are defined explicitly in the function body and renamed as *dat* and *window* using the code, "as<int>(k)" and "NumericVector dat(x)" for the vectors x and integer k respectively. Other variables required were declared within the C++ code. The length of vector *dat* was calculated using the function "*dat.size*", and a second numeric vector (*ret*) was defined to store the values that were the results of each iteration of the moving average loop. A float variable *"summed"* was defined to contain the calculations performed at each iteration.

Define *while* and *for* loop

The moving average was calculated for each element of *dat* until the *window* located at the end of the vector *dat* (length of *dat – window* +1) was reached, controlled by a *while* loop. An inner *for* loop generated the sum across successive windows of length *k* along vector *dat.* These data were stored in the parameter *summed,* and divided by *window* to give the moving average, which was stored in the vector *ret. Ret* is returned to R via the wrap function when the loop concludes. This converts the C++ data type (*ret*) back to an R object (an SEXP).

Compile, link and load function

The code described above was saved as a character variable in R, and the *cxxfunction* of the Rcpp package was used to compile the code written in C++, and to link and load it in R. This makes the function easily accessible in R, and also allows the function to be coded in C++ within the R environment.

Benchmark against R Moving Average

The performance of the moving average function written in C++ and called to R using the Rcpp package was compared against the given R function. The functions were both called using a randomly generated sequence of numbers and a window of 2. The function speed was returned using the command microbenchmark, from the microbenchmark package in R. Table1 below shows that the Rcpp function is superior to the R function in terms of speed of processing. In fact, the mean return time for the C++ function was 11 microseconds compared to 3892 for the R function.

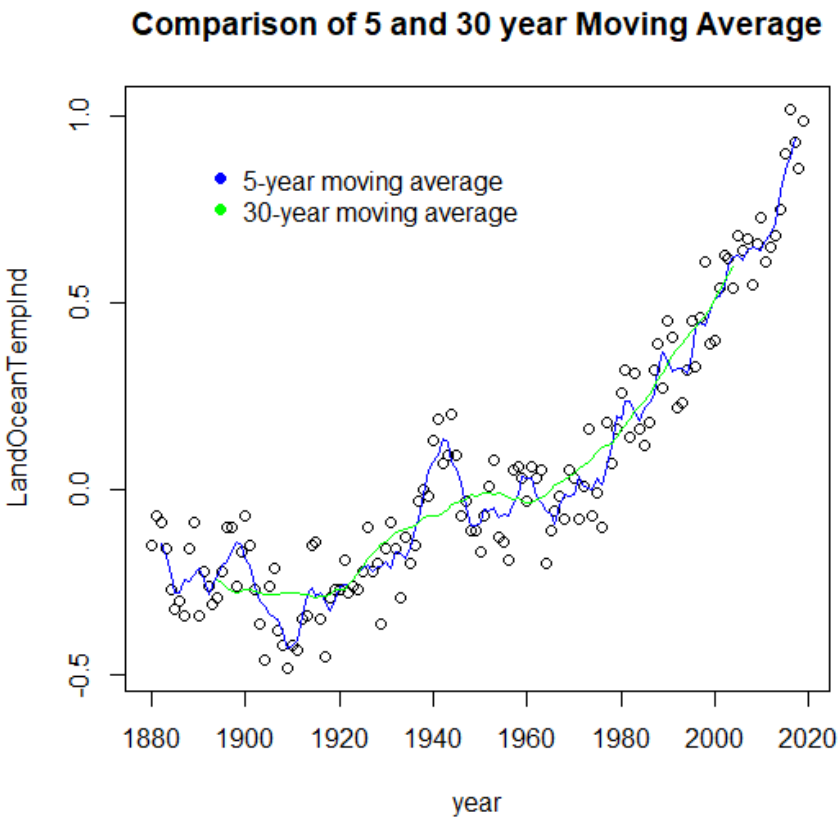
Table1: Comparison of time take to run moving average function in R (moving Average) and in a R function compiled from C++ code using the R package RCPP

Unit = microseconds	Mean	median
C++ Function called from R (rcppMovingAverage)	11.3	4.4
R function (MovingAverage)	3892.4	3895.2

The Rcpp function coded in parts (b) was used to calculate moving average ocean temperatures in five year windows (from 1882–2017) and 30-year windows (from 1894–2004). The results are illustrated in Figure 1. This plot shows how the 30year window smooths out variation over shorter time periods, to make the upward trajectory of temperature since 1920 more clear. The 30-year window removes

the effects of year-to-year changes in temperature that are of small consequence to the overall trend across the 140 year time range

Figure 1 The 5-year moving average compared to the 30-year moving average (green)



Implementing an Efficient Random Walk Algorithm in R

A function was written in C++ to simulate the journey that a tourist might take, given that they can only move one step in 4 directions, and the probability of any step is 0.25. The tourist started from (0,0). The code was implemented in R using the cxxfunction of the RCPP package. Function arguments passed from R to C++ were an integer that contained the number of tourists that were to be simulated using the model. The function consisted of a single for loop that iterated through the length of the "*tourist*" integer passed from R. A random number generator returned 0, 1 or -1 that was stored in a "*step*" variable to indicate the step taken at each iteration. The value for *step* was next used to update a vector that indicated the current position of the tourist along the x and y coordinates. An if statement was used to check if the current position matched the tourist office coordinates. If this was true, then an integer variable *"reach_dest"* that stored the number of times the tourist office was reached was advanced by one. This variable was returned to R at the end of the loop.

The efficiency of this code is optimised by writing the function in C++ and calling it from R, as opposed to using available R packages or functions. This should make the function run faster, which might be important for long simulations. The efficiency is further improved by ending the walk once the tourist office has been reached using the continue function in C++. This assumes that more than one visit to the office is not important, and gathering this information is wasting processing time.

A second version of the code developed in parts (a) and (b) was developed to address the presence of two addition tourist office sites. The individual probability of a tourist finding each office in a 10-step random walk is shown in Table 2. When a combination of the existing office or the new office was considered, the probability increases to 0.161 for Office 1 and 2 and 0.146 for Office 1 and 3. Although there seems to be an advantage to two offices as opposed to one, there is only a small difference between the two new offices and further simulations are necessary to establish if this is significant.

Table 1 Probability of finding Offices 1-3 in a 10 step random walk

Individual Probability			**Combined Probability**	
Office 1	*Office 2*	*Office3*	*Office 1 or 2*	*Office 1 or 3*
0.075	0.086	0.071	0.161	0.146